ASSEMBLER

External functions and procedures may be written in assembly language and assembled by the JRT assembler, as EXEC JRTASM. The assembler will then prompt for the input file name and options. Normally you will choose n.

JRTASM will then assemble the .ASM source in two passes and create a corresponding .INT file. The source contain standard 8080 mnemonics, but should not contain an ORG unless you intend to create a .COM file.

The following directives are allowed:

DB, DS, DW, ELSE, ENDIF, EQU, IF, ORG, READ, SET, WRITE

```
        WRITE 'Enter number of bytes to
reserve: '
n       READ
        DS  n
```

For external procedures or functions, the first five bytes must be

```
DB 95,6,0,92,0
```

On entry to an external procedure,

HL	is BASE (BASE + 6 is the origin for the global variables)
DE	is CUR (The length of the parameter block is at CUR-2. Arguments preceed CUR-2.) Return values for a function are placed (carefully) before the arguments.
BC	is TOS (The start of usable storage.)

CONVERTM

EXEC CONVERTM to convert Microsoft format relocatable files (.REL files) to JRT format .INT files. CONVERTM will prompt for the input file name.

LINKER

External procedures may optionally be linked in with a main program by EXEC LINKER. This is entirely optional.

Enter the main program file name. LINKER will then list the external procedures referenced by the program, with an * by those already selected for linking.
Your options then are

1 to 63	select the corresponding procedure
– 63 to – 1	de-select this procedure
100	select all procedures
– 100	deselect all procedures
0	end selection, begin processing

The old main program will be renamed .IN2, and the new main program will be created if there are no input errors. All files must be on A:.

Microsoft is a TM of Microsoft, Inc.

EXECUTION INTERRUPTS

To interrupt a program during execution, type a control-N.

Options are

E	print external procedure table
G	set line range for trace
L+ L–	line trace on/off
N	print current line and procedure
P+ P–	procedure trace on/off
R	resume program execution
S	print system status display
Z	terminate the program

To use line trace the programs should have been compiled with the $L or %LTRACE options.

ACTIVITY ANALYZER

Activan keeps records of program execution and prints histograms of program activity.

Invoke the activity analyzer by including $A on the EXEC command line, as EXEC PGM $A. Programs must be compiled with the $L or %LTRACE options.

Activan allows you to limit the range of lines to be included in the histogram(s).

Typing a control-A during execution allows you a number of options, including printing histograms, clearing the counts, changing the range of lines.

Options are

C	clear the line counters to zero
E	end the program
H	print histogram of activity
I	initialize line range and spacing
R M	run the program with Activan
W R	run the program without Activan
Z	terminate the program

any other display command list

Activan will allow a final histogram to be printed before returning to CP/M.

CP/M is a Digital Research, Inc. TM.

DEBUGGING

TRACING LINES, PROCEDURES, and FUNCTIONS

JRT Pascal can print the number of each line as it is executed. This is called line tracing, and requires that the program (or an appropriate portion) be compiled with the $L or %LTRACE option.

JRT Pascal can print the name of each function or procedure as it is entered and exited. This is called procedure tracing, and requires that the program be compiled with the $P or %PTRACE option.

Tracing can be turned on when the program is first executed by adding $L, $P, or $LP to the EXEC line, as EXEC PGM $LP.

Tracing can be turned on or off by interrupting the program with control-N and entering the appropriate options to the interrupt handler (L+ or P+ to turn tracing on and L– or P– to turn it off).

Tracing can also be turned on and off under program control by appropriate SYSTEM calls.

```
SYSTEM(LTRACE)
SYSTEM(NOLTRACE)
SYSTEM(NOPTRACE)
SYSTEM(PTRACE)
```

DEBUG procedure

Tracing can also be turned on or off under control of the external procedure DEBUG.

An external procedure, DEBUG, will prompt for line tracing options, limits of line tracing, and procedure tracing options.

Declare as PROCEDURE DEBUG; EXTERN;

Invoke DEBUG within the program with the statement DEBUG;.

JRT PASCAL

REFERENCE CARD FOR JRT PASCAL VERSION 3

Welcome to the expanding group of JRT Pascal users. You have joined a diverse group of business people, scientists, engineers, hobbyists, system developers, and software houses developing and using structured programs with our inexpensive, full featured PASCAL.

JRT Copy Policy

You are quite welcome to make copies of the JRT® Pascal disk and manual for personal friends. As a direct customer of ours, you are also welcome (even encouraged) to distribute the run-time package (EXEC.COM and PASCAL.LIB) with programs you develop for sale using JRT Pascal.

We do not give permission to resell JRT PASCAL.

Suggestions

Read (or skim) the manual first. Get a supplemental book on PASCAL. Decide whether you want to use JRT Pascal BEFORE you open the disks.

MAKE COPIES OF THE DISKS BEFORE USING THEM AND SAVE THE ORIGINALS.

Notation

In this reference card { } enclose optional items which may be repeated any number of times.

Asterisks indicate JRT extensions.

JRT Systems
45 Camino Alto, Mill Valley, CA 94941

RESERVED WORDS

abs
addr
allocate
and
array
begin
binary
boolean
call
case
char
chr
close
compress
concat
cons
const
copy
deallocate
delete
dispose
div
do
downto
else
end
eof
eoln
extern
false
file
fillchar
for
forward
free
function
get
goto
hex$
if
in
include
initialize
input
insert
integer
label
length
list
lrange
ltrace
map
maxint
mod
new
nil
nocons
nolist
noltrace
noptrace
not
nowarning
odd
of
open
or
ord
output
page
portin
portout
pos
pred
procedure
program
ptrace
put
rba
read
readln
real
real$
record
repeat
reset
rewrite
round
rrn
set
sqr
succ
string
system
text
then
title
to
true
trunc
type
until
upcase
var
warning
while
with
write
writeln
xor

BUILTIN FUNCTIONS AND PROCEDURES

	Name	Use
	ABS	ABS(integer__expression) ABS(real__expression)
	ADDR	* ADDR(variable)
p	ALLOCATE	* ALLOCATE(dyn__array__variable [subrange {,subrange}])
	ARCTAN	ARCTAN(real__expression) External function declared as FUNCTION ARCTAN (X : REAL): REAL; EXTERN;
p	CALL	* CALL(address, parameter__regs, returned__regs) Register order: flags, a, c, b, e, d, l, h
	CHR	CHR(integer__expression)
	CONCAT	* CONCAT(string__expr__1,..., string__expr__n)
	COPY	* COPY(source__string, position, length)
	COS	COS(real__expression) External function declared as FUNCTION COS (X : REAL): REAL; EXTERN;
p	DEALLOCATE	* DEALLOCATE(dyn__ array__variable)
p	DELETE	* DELETE(string__variable, position, length)
p	DISPOSE	DISPOSE(pointer__variable)
	EXP	EXP(real__expression) External function declared as FUNCTION EXP (X : REAL): REAL; EXTERN;
p	FILLCHAR	* FILLCHAR(structured__variable, length, character)
	FREE	* FREE (* Amount of free storage *)
	HEX$	* HEX$(any__variable)
p	INSERT	* INSERT(source__string, target__string, position)
	LENGTH	* LENGTH(string__expression)
	LN	LN(real__expression) External function declared as FUNCTION LN(X : REAL): REAL; EXTERN;
p	MAP	* MAP(pointer__variable, address)

...continued

	Name	Use
p	NEW	NEW(pointer__variable) NEW(pointer__variable, tag1,..., tagn)
	ODD	ODD(integer__expression)
	ORD	ORD(character__expression)
	PORTIN	* PORTIN(integer__expression)
p	PORTOUT	* PORTOUT(port__number, byte)
	POS	POS(pattern, source{, start__position})
	PRED	PRED(integer__expression) PRED(character__expression)
	REAL$	* REAL$(real__expression)
	ROUND	ROUND(real__expression)
	SEARCH	* SEARCH(table, key, count, param__blk); External function declared as FUNCTION SEARCH(VAR table:ar__rec; VAR key:key__type; ct:INTEGER; VAR p__b:search__parameter); EXTERN;
	SIN	SIN(real__expression) External function declared as FUNCTION SIN(X : REAL): REAL; EXTERN;
	SQR	SQR(real__expression) SQR(integer__expression)
	SQRT	SQRT(real__expression) External function declared as FUNCTION SQRT(X : REAL): REAL; EXTERN;
	SUCC	SUCC(integer__expression) SUCC(character__expression)
p	SYSTEM	* SYSTEM(COMPRESS) SYSTEM(CONS) SYSTEM(INITIALIZE) SYSTEM(LIST) SYSTEM(LRANGE, lower, upper) SYSTEM(LTRACE) SYSTEM(NOCONS) SYSTEM(NOLTRACE) SYSTEM(NOPTRACE. SYSTEM(NOWARNING) SYSTEM(PTRACE)
	TRUNC	TRUNC(real – expression)
	UPCASE	* UPCASE(string__expression)

EXTERNAL PROCEDURES

External procedures and functions are declared in the calling program, followed by EXTERN;

Individual external procedures (and functions) are written on separate source files as regular procedures or functions. The source file name must be the same as the procedure name and with the suffix .PAS.

External procedures start with the keyword EXTERN followed by optional CONST, TYPE, and VAR declarations for global variables and optional external procedure declarations. External procedures and functions are then written as regular procedures or functions. The external procedure must end with a period.

Global variables must be declared in the same order and have the same type as global variables in the main program.

External procedures called by an external procedure must be declared in the same order as those procedures are declared in the main program.

```
(* this is file getsize.pas *)
EXTERN
(* global declarations *)
CONST db__name__length = 4;
TYPE db__type =
ARRAY[1..db__name__length] OF CHAR;
VAR db__disk : CHAR;
    db__name : db__type;
(*getsize - return data base size *)
PROCEDURE setup(VAR dbn:db__type);
EXTERN;
FUNCTION getsize : INTEGER;
VAR s : INTEGER;
    f : FILE OF INTEGER;
BEGIN
setup(db__name);
OPEN(f,db__disk + ':dbf' + db__name + '.dat',BINARY);
READ(f, rba, 2; s);
CLOSE(f);
getsize : = s;
END; .
```

In this example, set up must be the first external procedure declared in the main program. Also, db__disk and db__name must be the first two variables declared in the main program.

STATEMENTS

Type	*Usage*
Assign	variable or function__name : = expression
BEGIN	BEGIN statement } { ; statement } END
Call	procedure__name { (expression { , expression }) }
CASE	CASE expression OF * expression { , expression } : statement; { expression { , expression } : statement; } * { ELSE: statement; } END
IF	IF Boolean__expression THEN statement { ELSE statement }
FOR	FOR variable__name : = expression TO (or DOWNTO) expression DO statement
GOTO	GOTO label__number
REPEAT	REPEAT statement { ; statement } UNTIL Boolean__expression
WHILE	WHILE Boolean__expression DO statement
WITH	WITH variable {, variable } DO statement

EXPRESSIONS

Operands: array__name[index__expression
{ , index__expression }]
constants
function__name { (arguments) }
pointer__variable^
pointer__variable^.record__field
record__variable.record__field
[set__expression]
variables

OPERATORS in order of decreasing precedence
NOT (* highest *)
* / DIV MOD AND
+ − OR
= <> > < > = < = IN (* lowest *)
Generally, parentheses are required with expressions combined with AND and OR.

INPUT – OUTPUT

CLOSE — CLOSE(file__variable)
REQUIRED for files that have been written.

EOF — EOF(file__variable)
Boolean function. For text mode input files, control-Z marks the end of a file. For binary input files, the end of the last block of 128 bytes is the end of the file.

EOLN — EOLN
EOLN(file__variable)
Boolean function. Indicates end of line when reading from console or text mode files.

ERASE — * ERASE(file__name)
External procedure: Erases a file from the disk. Must be declared as an external procedure.

```
PROCEDURE ERASE
(NAME:STRING[20]);
        EXTERN;
```

GET — GET(file__variable)
Procedure: reads another record from the file. Record length is determined from the file declaration. Get is often followed by one or more assignments, such as

```
rec__ord : = file__variable^;
```

Input modes — In text mode input, input items are delimited by spaces, tabs, commas, semi-colons, or carriage returns. Any characters not required to fill an input item are ignored. So

```
READ(single__character, char__6,
integ__er);
        OPTION SW 123
```

would return the single character 0, SW followed by 4 spaces, and 123. Console input is always in text mode (except for READLN (string__variable)).

In binary mode input from files, no translation is done. As many bytes are read from the file as is necessary to fill the input__list. Single characters are 1 byte, integers are 2 bytes, reals are 8 bytes, records are the sum of their individual members, and arrays are the number of members times the size of the members.

...continued

OPEN — * OPEN(file__variable, file__name, BINARY or TEXT
Procedure: opens file__name for random input and/or output. Name can be an array of characters, a string literal, or any string expression. File names can start with a drive letter a through p and a colon. The random record size is taken from the file__variable declaration.

Output modes — In text mode, scalar variables and expressions are converted to readable form. Structured variables are treated as arrays of characters. In binary mode, no conversion is done from internal form.

PUT — PUT(file__variable)
Procedure: writes another record from the file__variable buffer to the file, which must have been associated with a disk file with rewrite (in the binary mode). Put is usually preceded by one or more assignment statements, such as

```
file__ variable^ : = rec__ord;
put(file__variable);
```

READ — READ(input__list)
READ(file__variable; input__list)
READ(file__variable, RRN or RBA, expression; input__list;

READLN — READLN(input__list)
READLN(file__variable; input__list)
READLN(string__variable)
Reads a line of input from the console (up to the maximum width of the string variable). This is the only way to read input literally from the console.

RENAME — * RENAME(old__name, new__name)
External procedure: Renames a file on the disk. Must be declared as an external procedure.

```
RENAME (O, N: STRING[20]);
EXTERN:
```

RESET — RESET(file__variable, file__name, BINARY or TEXT, buffer__size)

...continued

REWRITE — REWRITE(file-variable, file__name, BINARY or TEXT, buffer__size)

SYSTEM — * SYSTEM(CONS) output to console
SYSTEM(LIST) output to printer
SYSTEM(NOCONS) no console output
SYSTEM(NOLIST) no printer output

WRITE — WRITE(output__list)
WRITE(file__variable; output__list)
WRITE(file__variable, RBA or RRN, real__expression; output__list)
Used with random files which have been OPENned. RRN indicates that expression is the record number in the file. Record numbers start at zero. RBA indicates that expression is a byte address within the file (starting at byte zero). The record or byte number expression can be an integer or a real number. Output will be translated to text or left in internal binary mode as chosen by the open/reset/rewrite statement.

WRITELN — WRITELN
WRITELN(file__variable)
WRITELN(output__list)
WRITELN(file__variable; output__list)

COMPILING PROGRAMS

Compiling programs or external procedures (or external functions).

[drive:]JRTPAS3 [drive:]file-name [$options]

Files required by the compiler

JRTPAS3.COM	The main program
PASCAL0.INT	Overlays required
PASCAL1.INT	
PASCAL2.INT	
PASCAL3.INT	
PASCAL4.INT	
PASCAL.LIB	Library, including error messages

A source program with filetype .PAS, such as PGM.PAS

Options

E	Stop when an error is encountered and allow compilation to be terminated.
L	Compile the program so line tracing is allowed.
P	Compile the program so procedure tracing is allowed.
Tch	Write compiler listing to: X - console Y - list device Z - no device A..P - 'LST' file on this disk

Examples

JRTPAS3 DEMO $TY
JRTPAS3 B:PGM $ELP
B:JRTPAS3 PGM

The compiler will produce a file with the filetype INT (such as PGM.INT) on the same disk as the source program.

EXECUTING PROGRAMS

[drive:]EXEC [drive:]file-name [$options]

Files required

EXEC.COM
PASCAL.LIB
A compiled program (such as PGM.INT)

Options

A	Allow histograms of program execution (Activan)
L	Start tracing line numbers
N	Perform an EXEC interrupt before executing the program
P	Start tracing procedure and function calls

Examples

EXEC B:PGM $AP
EXEC PGM where PGM.INT is on the current drive

IDENTIFIERS

Identifiers may be up to 64 characters long; all characters are significant. The first character must be alphabetic. Succeeding characters may be alphanumeric, the dollar sign, or the underline. All alpha characters are converted to upper case inside the computer.

NUMBERS

Integers range: – 32768 to + 32767

Hex integers format: suffix 'H'
leading '0' if first digit is alpha

3AH 0EADH 12FH – 0fffH

REAL floating point: 14 digits
exponent range – to + 63
Binary Coded Decimal format

The decimal point is required in REAL literals. The exponent is optional, but must be in fixed format if present: character 'E', sign, 2 digits.

3.14 0.0098 2.0E – 60 – 15.00138974e + 47

COMMENTS

Comments are enclosed in { } or (* *).

COMPILER DIRECTIVES

% INCLUDE ('filename')	insert a source file
% LIST	turn on compiler listing
% LTRACE	line trace codes
%NOLIST	turn off compiler listing
%NOLTRACE	no line trace codes
%NOPTRACE	no procedure trace codes
%PAGE {(lines/page)}	emit form feed
%PTRACE	procedure trace codes
%TITLE ('title')	display title in listing

PROGRAMS

A main program starts with PROGRAM program__name; Information in parentheses (input, output) is optional.

```
PROGRAM program__name;
declarations
BEGIN
{ statements; }
END.
```

A period is used at the end of a program.

PROCEDURES and FUNCTIONS

```
FUNCTION function__name{ (parameter__list ) }:
                          type__name;

PROCEDURE procedure__name{ ( parameter__list
                          ) };
```

The parameter__list is made up of one or more parameter__items separated by semi-colons. A parameter__item is

```
{VAR}parameter__name
    { , parameter__name}:
    type__name
```

Examples

```
FUNCTION    date : STRING[8];
FUNCTION    gcd(m, n : INTEGER) :
            INTEGER;
FUNCTION    getl (VAR  lin : STRING;
            maxlen: INTEGER):
            INTEGER;
PROCEDURE   cleanup;
PROCEDURE   randomize(r : REAL);
PROCEDURE   read__rec(VAR rec :
            record__type; VAR nr :
            INTEGER);
```

Function and procedure declarations can be followed by EXTERN, FORWARD, or the body of the function or procedure. The body consists of optional LABEL, CONSTant, TYPE, VARiable declarations and the BEGIN { statements; } END.

FORWARD declarations must be satisfied later in the source program by a FUNCTION function__name; or PROCEDURE procedure__name; followed by the body of the function or procedure.

DECLARATIONS

Type *Usage*

LABEL LABEL label__number { , label__number } ;
All labels used within the program must be declared.

CONST CONST const__name = literal character, string, integer, or real;
{ , name__2 = literal; }
Expressions such as chr(27) are not allowed as literals.

TYPE

```
TYPE type__name = type
{ , type__2 = type; }
Simple__types:
        BOOLEAN
        CHAR
        constant..constant
        INTEGER
        REAL
        (identifier{ ,identifier } )
Types:
        ARRAY[simple__type
            { , simple__type }
            OF type
        FILE OF type
        RECORD
            { member__name :
            type }
            {
            CASE { identifier : }
                type__name OF
            constant{ ,constant} :
                ( field__list )
            { constant { ,constant } constant
                ( field__list ) }
            }
            END
        SET OF simple__type
        simple__type
      * STRING
        { [maximum__length] }
        type__name
        ^type__name
```

VAR

```
VAR variable__name {,variable__name} : type;
    { variable__name { ,
    variable__name } :
      type; }
```

Subsidiary functions and procedures are declared after the VAR section. These functions and procedures have access to the variables declared in the outer procedure or function.